Pixel test beam visualization

Evan Chang
Argonne National Laboratories
ejchang@umich.edu

I. ABSTRACT

The inner detector of the large hadron collider (LHC) at CERN will be replaced before 2024. To effectively test the new hardware that will be implemented, a pixel telescope was used to test pixel sensor modules like those that would be used in the new inner tracker. Data was collected while running a test beam through the apparatus. I have generated a program that allows a user to visualize the event data, including hit pixels and reconstructed tracks. With this software, several errors were uncovered in the test beam tracking algorithm, which led to an improvement in the tracking efficiency measurements.

II. INTRODUCTION

Currently, the inner detector of the LHC at CERN contains silicon pixels, silicon strips, and a gaseous tracker. In 2024, this will all be replaced by a completely silicon tracking detector. There are several new pixel technologies that are candidates for use in the inner tracker that need to be tested. A pixel telescope was constructed using six pixel detectors using the technology from the current pixel detector. Two other types of detectors are placed in the telescope as the

devices under test (DUT's). One type: HVCMOS, and another: a pixel sensor that is similar to all the other telescope planes. The DUT that is similar to the telescope pixels uses a separate vendor for a delicate process known as bump bonding that connects the sensor to the readout electronics using a microscopic conductive ball. CMOS pixels have no bump bond between the readout electronics and the pixel sensor. The six telescope plane modules are to used to reconstruct a reference track to compare to the performance of the DUT's. A proton beam passed through the telescope to generate data. This data will be used to evaluate the performance of the DUT's using software that can interpret the pixel data, group the data into individual events, and provide information on each event. This paper is about the work that I have done creating a visualization program to be added to the pixel data software. The addition of this new feature helped detect errors in the tracking algorithm. It is also used as a tool for immediate validation of the alignment parameters and track reconstruction.

III. METHODS

To generate a three dimensional visual for a test beam event, it is critical to understand how the data from the detectors is generated. A hit occurs when a proton passes through the silicon creating electron hole pairs. The electrons are accelerated by an electric field towards an electrode. The electronics detect a spike in voltage, and if the voltage exceeds a given voltage threshold, the spike is registered as a hit. The duration of time when the measured voltage is above the threshold is called the time over threshold. Each event holds hit information from every sensor in the telescope. Multiple hits can be recorded on each detector for an event if multiple pixels record a hit. From these events, the tracking software generates cluster centers, which represent the estimated location of the proton when it crossed through the sensor. This is determined by taking an average of the "hit" pixels' location weighted by their respective time over threshold. If a pixel records a hit, the proton may have passed through the pixel at its edge or it may have passed through the pixel near its center. This leads to some uncertainty as to the proton's location. The tracking software attempts to reconstruct the path of the proton through the telescope as a line. Using the telescope geometry and the location of the hits, the software attempts to create a best fit line for the track. If the line has a chi-square value lower than a given amount, the track location is saved as a member of the event, along with the hits. Tracks are stored as an x_0, y_0, x-slope, and y-slope. The x_0 and y_0 are the coordinates of the track's z=0 intercept and the x-slope and y-slope represent x/z and y/z.

With the hit information and track data, it becomes possible to evaluate the performance of the pixels. For example, if a certain pixel is detecting hits at a very high rate, this pixel may be "noisy," and is thus faulty, since it is most likely detecting too many hits. Similarly, if a certain detector constantly detects hits that are consistently far away from the predicted value from the track, the readout electronics may be recording the hit in an incorrect location. These errors are easy to find by simply examining the data. However, other errors are much harder to find without any visual aides. This is why I developed a visualization program to view events. By being able to view three dimensional representations of events, it becomes very easy to look for faults in the hardware and tracking algorithm.

The visualization program is written in C++ and is run using a C++ interpreter. It works by using the ROOT geometry libraries, test beam event data, and sensor geometry files to create

a visual representation of the telescope. The program has several functionalities. It can display an event completely to scale, displaying the sensors, track, and highlighting and expanding the hit pixels, so every piece of event information is displayed.

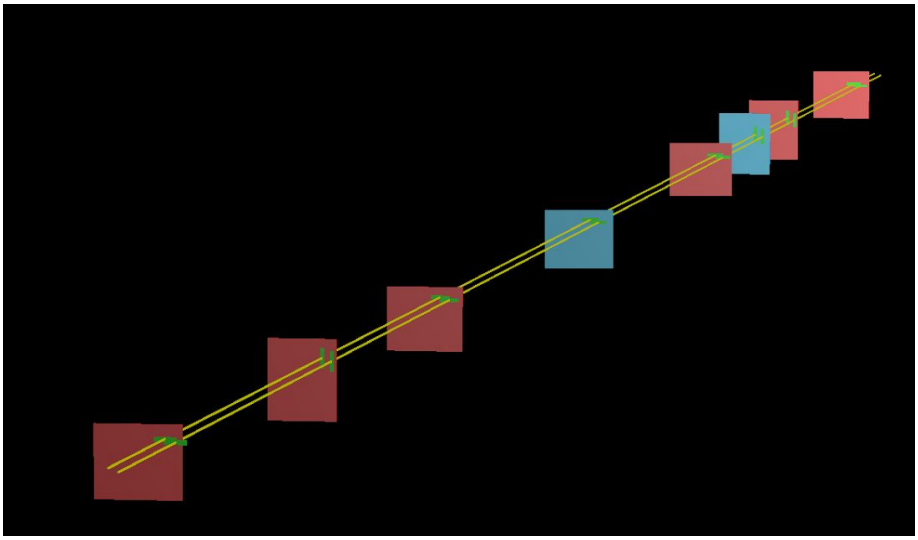


FIG. 1. Two tracks being displayed in a single event.

The above figure shows an event with two tracks detected. The telescope planes are in red and the DUT's are in blue. The yellow lines are the tracks, and the green boxes are the hit pixels being blown up to make them visible. The visualization program can also display a version of the event that only shows the track and the pixels that registered a hit. While the first functionality mentioned was initially all that I intended to create, I found that given the size of the pixels in relation to the entire sensor and to the size of the entire telescope, it is extremely difficult to find any possible errors, since viewing the pixels themselves is difficult at that scale. Thus, I developed a way to view the pixels with the z-axis scaled down and with only the hit pixels and the track displayed. This makes examining the pixels' locations with respect to the track and to each other much easier.

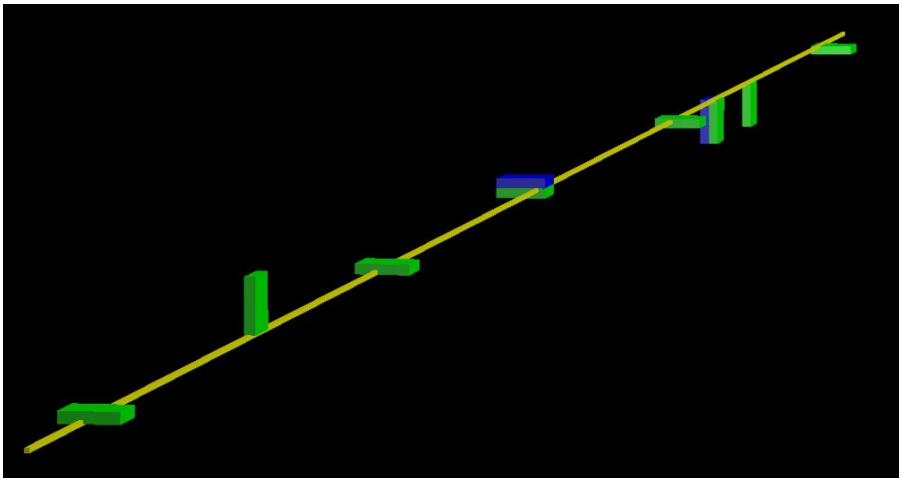


FIG. 2. Event displayed as only pixels and track.

The above figure shows an event only as the hit pixels and the track. The hit pixel that is indexed first by the tracking software is shown in green, while the second hit pixel is shown in blue. This allows the viewer to determine if multiple pixels were hit by the same proton, as two adjacent pixels with the same color may be difficult to distinguish. This simultaneous hit detection is often caused when protons pass on the border of two neighboring pixels, as shown in the two multi-hit planes in Figure 2. However, multiple pixels may also detect a hit if a proton collides with an election and scatters it parallel to the plane of the sensor, creating a delta ray.[1]

IV. FINDINGS

Several errors in the tracking program were discovered with the visualization program. An event that illustrates the problem is displayed below.

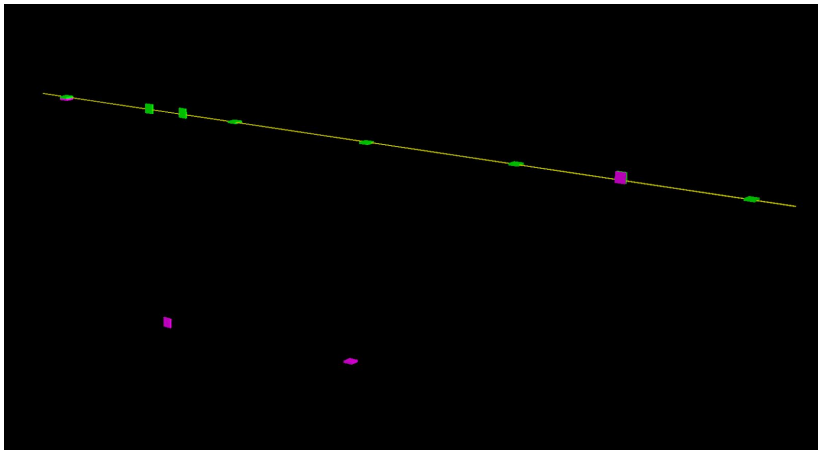


FIG. 3. Test beam event displayed as pixels and track showing error in distance variable.

[1] See Appendix for examples of unusual electron scattering

The magenta pixels in the lower left of the figure are hits recorded on the same plane as other pixels being shown. The code initially treated all of the pixels in this event as though they were exactly 1μm away from the track. This is clearly false, as there are some pixels that are clearly further away from the track than others. Furthermore, since each pixel is 50μm by 250μm and 250μm thick, the pixels that are far away from the track are obviously far greater than 1μm away. By examining the visualization of an event, this error is very easy to detect.

This error was actually discovered simultaneously by researchers from the Institute of High Energy Physics in Barcelona. This error in the code was fairly obvious, as a distance was being stored as a boolean instead of a double, causing all distance values stored with this variable to be one unit. While this can be found by painstakingly sifting through the code, it becomes obvious when using the visualization program alongside the code.

The visualization program was also useful in detecting biases in the test beam tracking algorithm. For example, The x_0 and y_0 of the tracks were being placed into a very spread out grid, rather than the tight grid that was expected. By looking at all the events in a test beam run in the direction of the beam, the tracks can very clearly be seen in an erroneous pattern. The tracks appeared to be in an array, yet the distance between the nodes of the array was larger than expected. Another bias discovered was found in the clustering algorithm. An absolute value function was missing from the cluster-track distances, and this was leading to tracks being placed incorrectly to one side of the pixels more than the other.

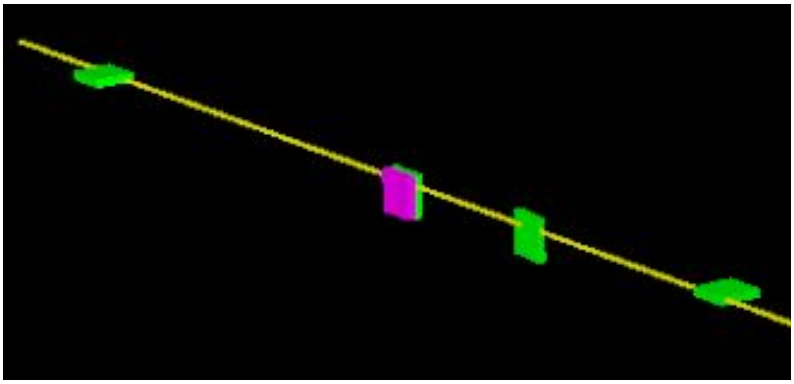


FIG. 4. Zoomed in test beam event displaying clustering bias.

The above figure is a track being rendered in the same way as Figure 3, except this figure is zoomed in to get a better view at what is happening in the third plane from the left, or the second pixel from the right. It is possible to see that the track is barely clipping the near side of the pixel. This in and of itself is not very noteworthy. However, this pattern was found in several other events, with virtually no events displaying this bias in the other direction. This was because of the missing absolute value modifier. Negative distance values were greatly skewing the residual values of tracks in which pixels laid to a certain side of the track, and this was ultimately preventing them from being seen as valid tracks. This led to the bias illustrated by the event above.

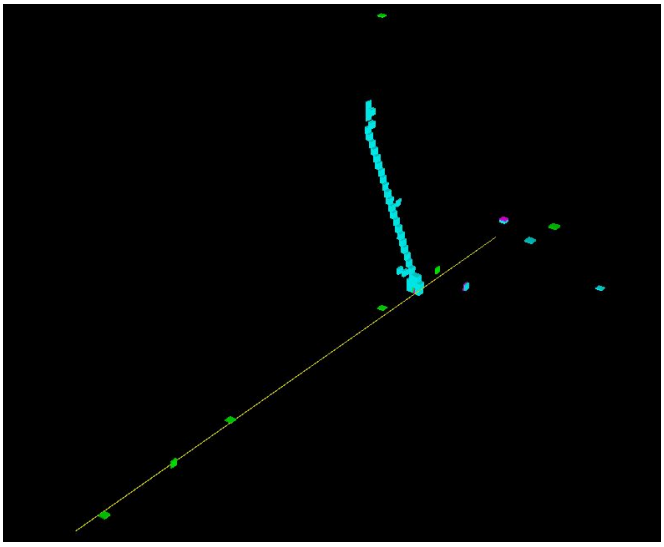
V. CONCLUSIONS

While this software was only used on one telescope configuration, it is useful for all test beam studies. The program is able to read geometry input files, which means that it can display events from any telescope and any pixel sensor, no matter how they are constructed. With my program integrated into the tracking software, it will now be much easier to quickly confirm the efficiency of the testing being done, search for more errors in the tracking algorithm, and quickly check that the geometry configuration files were done correctly.

VI. ACKNOWLEDGEMENTS

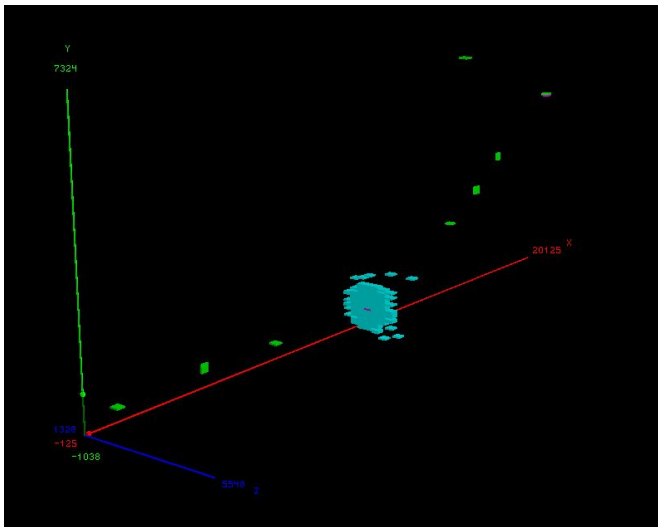
Thanks to Dylan Frizzell for assistance in combining my visualization with the Proteus software and discovering the errors in the test beam tracking algorithm. Thanks, as well, to Jessica Metcalfe and Sergei Chekanov for their guidance and assistance in testing my program.

VII. APPENDIX A



This event shows an example of a delta ray scattering across one of the DUT planes. The proton was clearly travelling from the left side of this image to the right, and the planes after the collision show a random arrangement of hit pixels due to further scattering. What makes this event even more interesting is that despite the noise and obvious lack of event quality, a track was still generated by the tracking software.

VIII. APPENDIX B



In this image, I included the axes[2] to display the change in the proton's path after the collision with an electron. The proton collides with a nuclei in one of the DUT planes and

[2] Using the axes as a straight line guide is all that matters. The units on the axes are skewed, since the image has been scaled down on the z-axis to make it easier to analyze.

clearly recoils in a different direction. The nuclear recoil causes a large scattering of electrons around the struck nucleus. This type of event is very rare and the only instance in hundreds of events that were searched manually.